RAGAM
Modern Application Development1 Project

Author:-
Anurag Sinha (213002198)
21f3002198@ds.study.iitm.ac.in

About me : A proud student of IITM and a passionate developer. I'm from Bihar and love programming, cycling and reading books.

Description:-

"RAGAM is a dynamic web platform enabling users to enjoy their favourite music seamlessly. Users can explore and listen to a vast collection of tracks while creators have the ability to upload and share their preferred songs. Admin oversight ensures the smooth operation of the platform.
Powered by Flask, Jinja2 templates, and leveraging SQLite, RAGAM provides a user-friendly interface for music enthusiasts, creators, and administrators alike, offering a seamless experience for music discovery and sharing."

Technologies used:-
1. Flask - As the main backend web framework.
2. Sqlite - Data query and schema design
3. Flask-Sqlalchemy - For easy management and integration of databases in flask app.
4. Flask-Login - For handling user authentication efficiently and securely.
5. Werkzeug-Security - For generating password hash
6. HTML, CSS, Jinja2 and Bootstrap - For front end development.
7. Matplotlib: Plotting library of python for creating visualizations like graphs.

DB Schema Design:-

- The database schema derived from the provided `model.py` includes five main tables: `User`, `Song`, `Album`, `Playlist`, and `PlaylistSong`.
- The `User` table contains fields `id`, `username`, `password`, `name`, `role`, `is_creator`, and `blacklist`. It's associated with playlists through a one-to-many relationship.
- The `Song` table includes details like `id`, `title`, `lyrics`, `duration`, `date_created`, `artist`, `file_path`, `genre`, `flag`, and `album_id`. It's linked to playlists through a many-to-many relationship via the `PlaylistSong` table. Additionally, it holds an optional `Album` foreign key relationship, and it calculates the `average_rating` property based on its associated `Rating` instances.
- The `Album` table holds `id`, `name`, `user_id`, `flag`, and is connected to songs through a one-to-many relationship, signifying that an album can contain multiple songs.
- The `Playlist` table comprises `id`, `name`, and `user_id`, establishing a one-to-many relationship with users. It's related to songs through the `PlaylistSong` table in a many-to-many relationship.
- Lastly, the `PlaylistSong` table serves as an intermediary for the many-to-many relationship between `Playlist` and `Song`, defining the associations between playlists and their respective songs.

- There's a `Rating` table that includes `id`, `user_id`, `song_id`, and `rating`, creating a many-to-many relationship between `User` and `Song`, allowing users to rate songs, with each rating being associated with a particular user and song

Project Layout:-

PROJECT

```
├── app.py
├── models.py
├── routes.py
├── Report.pdf
├── AdminCredentials.txt
├── instance
│   └── my_music_app.db
├── static
│   ├── uploads/
│   │   └──audio1.mp3
│   └── images
└── templates
    ├── acreatorprofile.html
    ├── admin_base.html
    ├── admin_dashboard.html
    ├── admin_search_result.html
    ├── adminlogin.html
    ├── aread_lyrics.html
    ├── aview_album.html
    ├── base.html
    ├── create_playlist.html
    ├── creator_profile.html
    ├── dashboard.html
    ├── delete_album.html
    ├── edit_album.html
    ├── edit_playlist.html
    ├── edit_profile.html
    ├── edit_song.html
    ├── index.html
    ├── lyrics.html
    ├── playlistsongread.html
    ├── profile.html
    ├── playlistsongread.html
    ├── register.html
    ├── search_result.html
    ├── song_by_genre.html
    ├── songs_and_albums.html
    ├── uploadsong.html
    ├── userlogin.html
    ├── uview_album.html
    ├── view_album.html
    ├── view_playlist.html
    ├── songs_and_albums.html
```

Note:

- To use the admin side, you can use username: admin and password:123
- I Have used same audio file to upload different songs, due to size limitation.

Video URL :-

https://drive.google.com/file/d/1WOQW0J9W_hvj7V9hVDnQts28YKXbMcig/view?usp=sharing